Interaction Distribution Network

Shun-Yun Hu

Institute of Information Science
Academia Sinica, Taiwan, R.O.C.

Abstract. Content Distribution Network (CDN) has been effective in accelerating the access and growth for web content such as web pages and streaming audio and video. However, the relatively static and bulky data CDNs are designed to serve makes them unsuitable to support latency-sensitive interactive streams such as network games or real-time conferencing. In this position paper, we describe the concepts for an Interaction Distribution Network (IDN), which supports small, interactive data streams in a scalable manner.
An IDN shares certain concepts similar to a CDN in that end-user clients connect to the closest serving proxies for accessing and updating interaction data packets. However, the key differences lies in the bi-directional nature of the interaction streams, and that the data streams may belong to a certain "interaction group," within which some additional processing on the data are possible. An IDN may support existing instant messenger (IM) services and Massively Multiplayer Online Games (MMOGs), while enabling new usage scenarios. We discuss the key challenges, potential solutions, and implications of IDNs in this paper.

1 Introduction

The Internet was designed to disseminate data packets to a large number of audience, and the invention of World Wide Web (WWW) has provided very effective means to distribute information. Subsequently, the serving of static content was augmented by dynamic (e.g., dynamic web pages) and streaming content (e.g., video and voice). As the size of Internet users increases, so have solutions to serve content to them on a large scale. Content Distribution Networks (CDNs) [12] have been designed to offload the requests from popular web-sites so that both the number of concurrent users and the quality of user experience can be improved.

However, in addition to the relatively static (and possibly bulky) content data, another important type of data are *interaction data*, which are bi-directional, and often come in short bursts of small packets, for the purpose to facilitate real-time human-to-human interactions. Such data, except in certain special cases, have yet found general solutions to support many millions of users. For example, although technically feasible with dedicated resources, it is still uncommon for thousands of people listen to a *live* online speech, while asking live questions afterwards. Or, in emergency diaster reliefs, to gather groups of rescuers and

volunteers in groups up to hundreds or thousands, to brief them online for the tasks ahead, or to discuss quickly what to do (currently this is only feasible by a physical gathering within a big hall).

Even though Internet services such as instant messenger (IM) networks have been deployed for many years, where a user can send real-time messages to one or a group of contacts. Their usage scenario is limited to text or voice chat messages with a few other users (e.g., most IM sessions are between two persons, and a practical Skype discussion group is generally under 10 people). Interaction groups with a large group size have yet appeared on the Internet.

In recent years, Massively Multiuser Online Games (MMOGs) [1] have spawned out a new generation of Internet applications, where millions of concurrent users are joining in the same virtual world, to interact with one another in real-time. MMOGs provide a glimpse as to what future online interactions might look like, as they provide settings similar to what we experience in the real world, with spaces and gathering places. However, due to limited client-side bandwidth and server-side resources, interactions among a single group of people generally are limited to below 100 users. On the other hand, we are interested in interactions that may allow thousands or more participants, to hold events similar to real-world graduation speeches, holiday parades, rock concerts, and even political rallies.

We observe that while disseminating content (e.g., web pages, audio and video streams) is very matured, the dissemination of interaction (e.g., chat, online editing, gaming actions), have yet found a general mechanism for large-scale dissemination. In this paper, we term a mechanism that can serve this purpose as an *Interaction Distribution Network* (IDN), and describe why an IDN may allow new applications to develop, and help to scale existing interactions on the Internet. We describe an initial proposal for how an IDN may be implemented, and discuss some of its key issues and application scenarios.

2 Background

2.1 Content Distribution Network

CDNs [12] have been put in place soon after WWW took off. The main problem CDNs try to address is the *flash crowd* effect of popular content. For example, when a significant news event just happens, an online news site may suddenly receive traffics several times over the normal. In such case, a single service point would quickly be overwhelmed and ceased to provide services.

CDNs try to replicate the content onto several *edge servers* that could serve similarly as the main site to potential visitors. A request sent to the main server will be redirected to one of the *best* edge server so that the loading on the main site is distributed. The best selection may be based on a number of different criteria, such as latency experienced by the user, content availability, or the current load of the edge server.

Note that because CDNs serve relatively static content, the key issues in CDNs are content placement and request re-direction. Maintaining latency guarantee is not of primary concern (e.g., it is acceptable for a video stream to have a couple of seconds of start-off delay, as long as the bandwidth is enough to smoothly stream the video subsequently), nor is supporting potential processing on the data stream (e.g., as needed by MMOGs to update game states).

2.2 Interest Management

To support scalable user interactions in MMOGs, a basic approach is to limit the amount of data sent to each user. Ideally, only data most relevant to the user's current interaction should be delivered. The most basic interest management thus is for a server to gather all data streams, then filter the data individually for each user. For example, in a MMOG, the server may only deliver messages on events occurring within the receiving user's view, or the user's *area of interest* (AOI) [10].

To scale up such function, a common approach is to partition the virtual environment spatially into regions, and assign each region a channel (i.e., either a physical or application-layer multicast address). A user would subscribe to the channels that represent or overlap with its AOI, in order to receive only the relevant information. While such a basic technique would work, how to partition and load balance the virtual space then becomes the main challenge. Due to the often uneven user distribution across geographic or time of day [7], the proper partitioning becomes the major challenge in keeping the approach scalable yet economical.

2.3 Publish / Subscribe

To realize interest management, a common method is via the publish / subscribe (pub/sub) model, of which there are two main types: channel-based and content-based [2]. Channel-based pub/sub allows any user subscribed to a channel to receive messages sent to the channel. For example, chatrooms can be realized by having all users subscribed a given chat channel, and publish their respective chat text to the channel. Content-based pub/sub on the other hand, would deliver messages only if the message content matches certain pre-specified criteria (called *interest expressions*) specified by the subscribers. For example, for messages that have x and y coordinates, a user with a subscription request of $[100 < x < 200]$ and $[250 < y < 450]$ would only receive messages with coordinates that fall within the specified range.

A useful form of content-based pub/sub is spatial publish / subscribe (SPS) [4]. In SPS, a user specifies only a subscription area (on a 2D space), other users may then publish messages to either a point or an area. Only if the published area overlaps with the subscribed area(s), will the subscriber receive the message. Additionally, each subscriber can move its subscribed area continuously,

which helps to automatically receive the most spatially relevant messages, without having to query continuously. For spatial interactions such as those within MMOGs, SPS can be a more useful primitive than channel-based pub/sub.

3 Interaction Distribution Network

3.1 Basic Definition

We now describe what constitute as an IDN, and highlight its main characteristics. In contrast to a CDN, an IDN should serve interaction data instead of content data. By interaction data, we mean small packets that are generated by human operators, which can come in short bursts, and need to be delivered to intended targets in sub-second time-scale (e.g., the results of a user's action should reach other users in 250 milliseconds for an online action game). Another requirement is that the number of interacting entities within a group should be scalable, potentially in the range of thousands of more.

Existing real-time interactions on the Internet include IM networks, MMOGs, and some forms of collaborative editing (e.g., Google Docs that allow sharable online documents). When the supportable number of concurrent users are high, the infrastructures used can be seen as specialized instances of IDNs (e.g., large IM networks supporting tens of millions of concurrent users). However, an ideal IDN should be able to serve a generic audience, regardless of application types, much like how CDN can serve any web content (be it files or streaming media). Another important aspect is that an IDN should keep the interaction latency bounded within certain reasonable limits, while delivering only relevant data to users.

In terms of functionality, IDN should support a general data path, which can be best summarized as an "event → processing → update" path. We define the messages generated by the human operators as *event*, which are delivered (via the IDN) to some *state managers* for *processing*, according to application-specific rules, the results are then delivered to users who may be interested or are affected by the changes via *update* messages. For IM services, the *processing* stage is missing (unless for backup or analysis purpose at the server), and the messages only go through an "event → update" data path. However, MMOGs are more general and do indeed need the *processing* stage for calculating new game states based on rules of the game (i.e., the *game logic*). So existing data path in MMOGs can serve as good examples of the full interaction data path for an IDN.

We summarize some key requirements as follows:

Interactivity For any given interaction, there exists an *interaction group* where bi-directional communications are possible and are of mutual interest to the participants. The bi-directionality distinguishes *interactions* from the familiar *broadcasting* (e.g., Internet radio or YouTube streaming). Because interactions are between human participants, the delay between the generation of events and the receival of updates should be within application-specified limits. This is

mostly application-specific, for example, the latency requirement for voice communication should be less than 100ms, but for IM text it can be a couple of seconds.

Scalability There are two types of scalability: scalability in terms of the total number of concurrent users (which may be termed *system scalability*) and another is scalability within the same interaction group (which may be called *group scalability*). So while existing IM services can already scale to tens of millions of concurrent users (i.e., achieving system scalability), the number of participants is at most a few tens, and we do not yet see thousands of people within the same interaction group. A similar situation exists for MMOGs, where even though the size of the concurrent users in a game may be a few millions, each interaction group is often less than 100 people.

Generality While existing architectures may already support some form of IDN (e.g., IM networks and MMOGs), we consider that they are not yet generic enough to support different application types *on the same infrastructure*. This requires a generic design such that not only existing IM or MMOG services can be deployed, new classes of applications can also be enabled (e.g., large-scale interactive online lectures or rallies).

To compare with existing online interactions: an IDN is different from instant messenger services in that in addition to the simple "sender → receiver" data path, some optional processing can be placed on this path, enabling a complete "event → process → update" cycle. On the other hand, while current MMOGs already support such data path, they are not yet scalable enough so that thousands of people can be inside the same interaction group. In other words, although we already have scalable solutions for content distribution (via CDNs), we do not yet have scalable solutions for interaction distribution, thus the need for new designs.

3.2 Key Challenges

The requirements for an IDN can translate to certain challenges:

Bounded Delay (Interactivity) To support interactions, the key here is that the interaction delay should be bounded, though not necessarily minimized. In order to provide an upper bound for interaction delays, the time spent on the data path from the sender to receiver of an interaction message should be predictable and controllable, regardless of the current scale of the system or the interaction group. As the data path consists of event → processing → update, all three main stages should have predictable durations. While it is easy to control such latency in small group interactions (e.g., we can assign one available server to handle the message relay between two IM participants), it becomes more problematic if the interaction group size is large, and if processing is involved on the event messages. The main challenge then is whether the system can deliver to a large interaction group, while keeping processing delay within certain limits.

Load Distribution (Scalability) In order for the interaction to scale, some form of load distribution may be needed. This can be done in the form of having

proxy servers, where users can connect to the closest proxies to avoid over-connecting a single server, or in the form of having multiple processing servers (i.e., multiple state managers) so that the system is not brought down by overloaded processing. In general, both CPU and I/O operations on a IDN should be distributed across the available IDN nodes as to maximize scalability.

Dynamic Grouping (Generality) For any given interaction, an *interaction group* is formed among the participants, where members of the group can communicate with one another (pending some intermediate processing). In order to allow for a generic architecture, the membership of the group should be able to adjust dynamically based on application requirements. For example, new members can join and old members can leave for an IM discussion group, or users can walk in and out of an on-going battle field in an MMOG. While the rules of the group formation may be application-specific, interactions within the group should still adhere to the interactivity and scalability requirements mentioned above.

3.3 A Potential IDN Design

Recent research on scalable MMOG systems have shown that by separating different tasks traditionally located on a single server, group scalability may improve. For example, Lake et al. [6] show that by separating the network I/O function of a MMOG region server into a separate component (called a *client manager*, which is responsible to handle all incoming and outgoing traffic with user clients), they are able to scale the number of users in a Second Life region (256m x 256m) from 100 to 1000. In works such as DarkStar [11] and Najaran and Krasic [8], it is shown that if the function of message filtering (also called *interest management*) is made into a separate component from normal processing tasks, then message processing may become more scalable. In other words, by separating *interest management* from *state management*, we can improve the scalability of the system by: 1) lessening the load on state managers, and 2) utilizing more state managers in parallel to improve overall scalability when the processed tasks are independent from one another. We thus suggest the following components in a generic IDN:

Proxies: These are servers which the user clients directly connect to, and function much like web proxy or the edge servers in CDNs (e.g., the *client manager* component as proposed by Lake et al. [6]). A connected client will have a user instance created at the proxy to participate in the IDN on behalf of the user. Users should connect to physically closest proxy to minimize latency. Connections among proxies and between proxies and other servers are assumed to be high-speed. This way, overall latencies on the data paths can be reduced.

Matchers: These are servers that perform *interest management* for the user clients. So users (via their proxy instance) can specify interest subscription or requests to the matchers, who will then perform the necessary filtering of unwanted messages and deliver only messages with relevance back to the user proxies. The matchers can simply support channel-based pub/sub functions (i.e., all users subscribed to a given channel will receive messages sent to that channel, for

example, in Najaran and Krasic [8], a *Scribe* P2P overlay is used to perform channel subscription on a per-user basis), or spatial publish / subscribe (SPS) functions [4] (i.e., subscriptions and publications are specified as areas that can move with users, which is more suitable for MMOG-style interactions).

Managers: These are servers that perform *state management*, which is also the *processing* part in the "event → processing → update" data path. The events generated by users are used as inputs to further calculate and refine certain object states, based on the input events and some pre-defined rules. The results of the calculations (in the form of updated state variables) may then be communicated back to the users via the matchers and proxies.

From another view, proxies are the system's entry-points, and are representatives acting on users' behalf to participate in the IDN, for security and latency-reduction purposes; matchers form a communication layer, where messages within the IDN are passed; while managers form a computing layer, where tasks that require further computations (e.g., game state and logic calculations) are done.

The basic *"event → processing → update"* data path may then be realized in such a manner: a user-generated *event* is first sent to the user's proxy, which is then delivered to the respective matcher currently handling the interaction group(s) of the user (e.g., a chatroom in case of IM service, or a small region in case for a MMOG). If the event requires processing, for each interaction group, there is a respective manager subscribed to the group, in order to receive any events sent to the group. If the interaction group is large, it can be partitioned and handled by multiple managers. A manager may also query relevant states from other managers, in order to perform the correct *processing* (e.g., a manager that handles a given region in an MMOG may want to know the status of other users who reside in a neighboring region). Once the states are modified by the manager, the manager would publish the *update* messages to the matchers, who will deliver the updates to the respective subscribers (i.e., proxies on behalf of their users). The proxies then deliver the updates back to the users, completing the data path. In other words, the data path will look something like: user A → proxy A → matcher A → manager → matcher B → proxy B → user B.

It should be noted that when the processing stage is involved, only managers are allowed to subscribe for the user events, and user proxies can only subscribe for update messages. So the matchers responsible for delivering events (to managers) and updates (to users) are logically different, and can be hosted on different physical machines. Note that even though the data path involves six network hops, four of which are within the server cluster, which is assumed to be connected via high-speed networks. As the two external hops (i.e., between users and proxies) are also assumed to have low latencies (users should connect to their closest proxies), overall latency can be controllable.

3.4 IDN Variations

The above outlines the basic designs for an IDN, however, variations exist that can either improve performance or make IDN more customized to specific appli-

cations. For example, existing IM networks can be supported with only proxies and matchers, as chat messages generally do not require further processing (unless real-time analysis is required). A scalable channel-based pub/sub mechanism is sufficient for running the matchers. To support group communication, a chat channel is set up for all participants to join, each message sent to the channel is then delivered to all active participants (i.e., subscribers) of that channel. In another variation, the roles of proxies and matchers may be combined or run on the same physical machines. This will have the interesting effect that events in an MMOG that require processing can be separately handled from those that do not. For example, when the proxies have received a trade or attack event message (which may need to be processed based on current game states and game logic), the message should be forwarded to the manager. However, if the message is simply a movement or a chat, the matchers can deliver the messages directly to relevant proxies, without going through the managers, thus saving both processing and delivery time. If the proxies and matchers reside on the same machines, delivery can be quite efficient (e.g., user A $\rightarrow$ proxy A $\rightarrow$ proxy B $\rightarrow$ user B), given the assumed low latencies among proxies, and between users and their proxies.

4 Discussions

One important question is how interaction groups up to thousands of participants can be supported, while keeping delay bounded. The answer depends on both mechanism and hardware performance. For mechanism, the simplest delivery is based on channel-based pub/sub, where a given channel is hosted on a certain matcher, and all potential users interested in group interactions send subscription requests to the matcher. Whenever a message is sent to the matcher, the matcher will forward it to the subscribers. This mechanism faces a limitation when the subscriber size is large and beyond a single matcher's upload capacity. A simple extension is to allow the matchers to first forward the messages to some helper matchers, which can then forward the message to subscribers on their own. Certain application-layer multicasts such as Scribe (as used by Najaran and Krasic [8]) or VoroCast [5] can be used for this purpose. While there is latency penalty for the forwarding among the matchers, the cost should be small as it occurs within a LAN. As each additional helper matcher contributes its own processing capability, overall delivery time can be improved for many recipients. Whether the overall latency can be bounded is determined by the depth and speed of the forwarding among matchers. Actual hardware limit and latency thus will determine the maximum size of an interaction group.

Another important question is how to ensure that the processing time at the managers is also bounded. We note that for state management, the most generic form is that some existing object states are updated based on some new events and pre-existing rules. For example, the processing of a trade event from user A to user B in an MMOG would check the following: 1) whether user A has the trading item (e.g., money); 2) whether user B has the traded item (e.g., some

clothes); and 3) whether user A has enough space to carry the new item, etc. Such a transaction requires the knowledge of the game states of both user A and user B, as well as certain rules regulating the transaction (e.g., user A must have enough money and space for the trade to be successful). Afterwards, if the trade is successfully concluded, both user A and user B should be notified of the new states. We note that in such case, each transaction is quite localized and requires only a few states. It is thus possible to conduct many such interactions in parallel, independent from each other. If there are conflicting updates to a certain state, existing consistency techniques such as locking or synchronization model (e.g., primary-replica objects) can be employed. The key to scalable processing thus lies in ensuring that each individual state managers can have access to the relevant states and events in time [11].

Assuming that the update rules are already stored at the managers, the relevant question then is whether existing object states can be accessed within a given time constrain. This involves certain query operations of either key-value queries (e.g., access the object states for user 'John'), or spatial queries (e.g., find all objects within a 3 meters range that this rocket might collide). We note that key-value lookup can be done in $O(log\ n)$ time, where n is the number of searchable objects (e.g., using CAN [9]). While spatial query can be done within constant time (by using geographically partitioned overlay such as a VON [3] or on extensions of CAN). As long as the query time can be independent from the total number of objects searchable, or at least those key-value queries (which takes $O(log\ n)$ time) are few for a given event, the processing operations can spread out on different processors, independent of where the event takes place (e.g., as used by Project DarkStar [11]). State processing thus may become parallelizable on a massive scale.

In summary, the keys to support scalable interactions within the same group are: whether the message delivery and event processing time within the IDN can be bounded. For the former, we consider that using multiple matchers for forwarding will distribute messages to a larger set of subscribers, while incurring small latency overhead within the IDN's high-speed LAN environment. For the latter, we consider that as long as the query time for accessing existing object states can be constant, and that event processing generally involves only a few objects, it is possible to process a large number of events in parallel such that the processing is distributed on multiple machines and finished within a bounded time.

5 Application Scenarios

While webcast already enables up to hundreds to thousands of people to watch a live or recorded event on video over the Internet (delivered via CDNs), we do not yet see scenarios where the audiences can ask live questions afterwards and still be heard by all other participants. This can be seen as an extended form of a large chatroom, except that the number of participants may be too large for a single server to deliver all out-bound messages.

With an IDN, the speaker's voice or video will go through the speaker's proxy first, then it will reach the one of the matchers responsible for the speech publication. The first matcher could then forward the data stream to a number of helper matchers, which then could forward the speech stream to the subscribing users' proxies. The depth of the internal forwarding can be adjusted according to the number of total participants (e.g., the more participants, the higher the depth and width of the forwarding). The data stream finally will reach the participants via their respective proxies.

If any participant wishes to ask a question afterward, his or her data stream will go through exactly the same procedure as the speaker's (i.e., asker → asker's proxy → asker's matcher → helper matchers → audiences' proxies → other audiences). The only problem that will prevent the system to scale is if too many users are all speaking simultaneously. However, given that only one participant will be asking questions at a time in this scenario (provided that a moderator exists), the situation should be scalable to thousands of users.

6 Conclusion

This paper presents the concept for an Interaction Distribution Network (IDN). Although aspects of IDN already exist in today's instant messenger networks and MMOG systems, current systems can still improve in their group scalability and generality. We show that by integrating the separate components into a coherent design, scalability under bounded delay may become possible. The main difference between an IDN and existing systems is the separation of interest management from state management, so that message delivery and processing time may be bounded. We identify three components at the server side as proxies, matchers, and managers. Not only existing systems can be supported by using a combination of these components, new classes of applications can become feasible from the improved scalability and generality. While we have identified the basic IDN layout and how we can approach to build one, works remain on actually constructing one. This will provide a step towards supporting truly dynamic interactions on a massive scale on the Internet, and possibly new types of applications.

References

1. Alexander, T.: Massively Multiplayer Game Development. Charles River Media (2003)
2. Bharambe, A.R., Rao, S., Seshan, S.: Mercury: A scalable publish-subscribe system for internet games. In: Proc. NetGames. pp. 3–9 (2002)
3. Hu, S.Y., Chen, J.F., Chen, T.H.: Von: A scalable peer-to-peer network for virtual environments. IEEE Network 20(4), 22–31 (2006)
4. Hu, S.Y., et al.: A spatial publish subscribe overlay for massively multiuser virtual environments. In: Proc. 2010 International Conference on Electronics and Information Engineering (ICEIE 2010). pp. V2–314 – V2–318 (2010)

5. Jiang, J.R., Huang, Y.L., Hu, S.Y.: Scalable aoi-cast for peer-to-peer networked virtual environments. In: Proc. ICDCS Workshop Cooperative Distributed Systems (CDS). pp. pp.447–452 (2008)
6. Lake, D., Bowman, M., Liu, H.: Distributed scene graph to enable thousands of interacting users in a virtual environment. In: Proc. NetGames 2010 (MMVE session) (2010)
7. Lee, Y.T., Chen, K.T.: Is server consolidation beneficial to mmorpg? In: Proc. IEEE Cloud (2010)
8. Najaran, M.T., Krasic, C.: Scaling online games with adaptive interest management in the cloud. In: Proc. NetGames 2010 (2010)
9. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network. In: Proc. SIGCOMM (2001)
10. Singhal, S., Zyda, M.: Networked Virtual Environments. ACM Press (1999)
11. Waldo, J.: Scaling in games & virtual worlds. ACM Queue (Nov/Dec 2008)
12. Wang, L., Park, K., Pang, R., Pai, V., Peterson, L.: Reliability and security in the codeen content distribution network. In: Proceedings USENIX Annual Technical Conference (ATEC '04) (2004)